SC - 20/5/2025

20/05/25 10:08

ALCUNE APPLICAZIONI DEGLI INTERRUPT

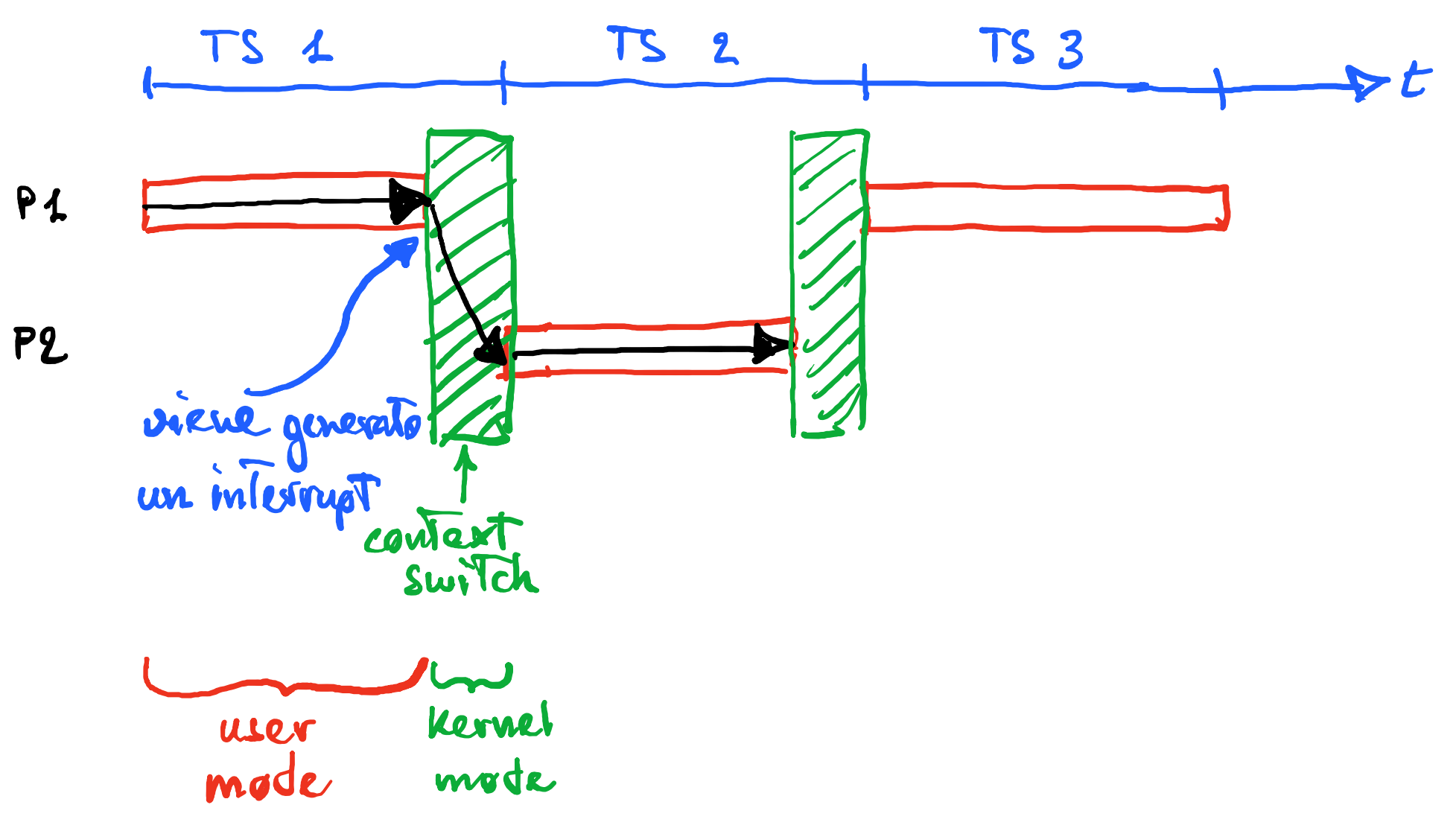
CONTEXT SWITCH CON PREEMPTION

Prevede Tre passi:

① La stato del processo interrotto viene salvato nel suo PCB

② Il PCB del 〃 〃 viene messo nella coda dei processi READY

③ Lo stato del processo READY che deve ora essere messo in RUNNING viene ripristinato nella CPU a partire dal suo PCB



SYSTEM CALL

In IA-32 una specifica Trap viene usata per invocare dal codice utente (user mode) una system call (kernel mode)
Usiamo l'istruzione ASM

```
INT 0x80   (o anche INT 128)
```

per richiedere l'esecuzione di una syscall al kernel.

Come passiamo i parametri alla syscall?

eax → identificativo della syscall da eseguire
ebx → 1° parametro
ecx → 2° 〃
edx → 3° 〃
esi → 4° 〃
edi → 5° 〃

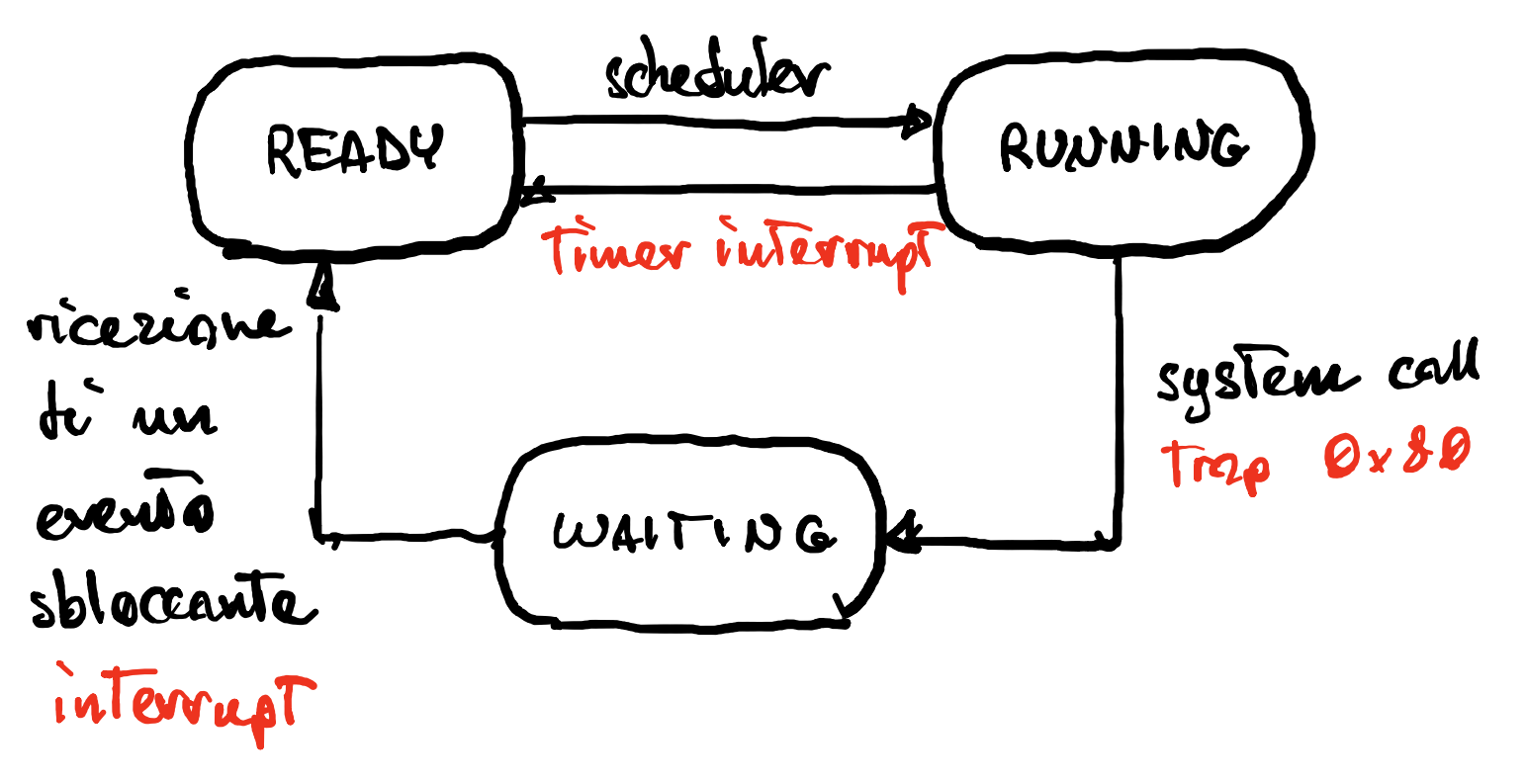
Il valore restituito dalla syscall (se previsto) è in eax

Esempio: implementiamo la funzione exit().

```
.globl my_exit

my_exit:
    pushl %ebx
    movl $1, %eax          // $1 → syscall exit
    movl 8(%esp), %ebx     // codice di Terminazione del processo
    int 0x80
    popl %ebx
    ret
```

PASSAGGIO DI UN PROCESSO DA WAITING A READY



SEGNALI

Un segnale è una notifica asincrona inviata dal SO a un processo.

	DESTINATARIO	MITTENTE	GESTORE
INTERRUPT	KERNEL	periferiche, CPU, programma	INTERRUPT HANDLER
SEGNALI	PROCESSO	kernel, programma	SIGNAL HANDLER

Ogni segnale è caratterizzato da:

- NUMERO IDENTIFICATIVO : identifica il Tipo
- PID : identifica il processo destinatario

Quando un processo viene notificato dal SO dell'arrivo di un segnale possiamo avere Tre comportamenti:

① TERM / CORE : il processo viene Terminato. Opzionalmente viene generato un "core dump"

② IGN : il processo ignora il segnale

③ handler : il processo esegue un signal handler precedentemente configurato.

I segnali principali:

CAT	SEGNALE	NUM	DESCRIZIONE	IGN?	GEST?	DEFAULT
TERMINAZIONE	SIGTERM	15	soft kill	✓	✓	TERM
	SIGINT	2	generato CTRL+C	✓	✓	TERM
	SIGKILL	9	hard kill	✗	✗	TERM
	SIGCHLD	17	Terminazione proc. figlio.	✓	✓	IGN
MEM	SIGSEGV	11	segm. fault	✓	✓	TERM
TIME	SIGALRM	14	Timer di sistema	✓	✓	TERM

ATTENZIONE: un interrupt, attraverso il suo handler, può generare un segnale indirizzato ad un processo.

Es.: CTRL+C → interrupt Tastiera → interrupt handler → SIGINT
↑ utente ↑ periferica HW ↑ kernel ↑ va al processo destinatario

CTRL+Z → 〃 → 〃 → SIGSTOP

Timer HW → interrupt Timer → 〃 → SIGALRM

GESTIONE DEI SEGNALI

Per inviare un segnale ad un processo:

```
#include <signal.h>

int kill (pid_t pid, int sig);
```

int ↑ [0; -1]
sig → identificativo del segnale
pid → PID del destinatario

ATTENZIONE: è possibile inviare un segnale solo ai processi dello stesso utente.

Anche da Terminale:

```
kill -9 12345
```

-9 ↑ num segnale
12345 ↑ PID destinatario

Per gestire un segnale:

```
int sigaction (int sig,
               const struct sigaction * act,
               struct sigaction * oact)
```

int ↑ [0; -1]

- int sig → il segnale da gestire
- const struct sigaction * act → indica come gestire il segnale
- struct sigaction * oact → ottenere info sulla attuale configurazione del segnale

```
struct sigaction {
    void (* sa_handler) (int),
    void (* sa_handler)(int, siginfo_t *, void *),
    sigset_t sa_mask
    int sa_flags
    void (* sa_restorer )(void)
}
```

- void (* sa_handler) (int) → come gestire il segnale:
  - puntatore a funzione (handler)
  - SIG_IGN
  - SIG_DFL
- void (* sa_handler)(int, siginfo_t *, void *) → usato per configurazioni più complesse (alternativa al 1° param.)
- sigset_t sa_mask → set di segnali che devono essere ignorati durante l'esecuzione dell'handler
- int sa_flags → flag che definiscono il comportamento del segnale
- void (* sa_restorer)(void) → deprecato (non utilizzato)